Tycho 2 User Guide (Draft)

Kris Garrett

September 22, 2016

1 Introduction

Tycho 2 is a reworking of a code named Tycho written by Shawn Pautz around the year 2000. The original code solved a linear, kinetic, transport equation on an unstructured, 3D tetrahedral mesh and was parallelized via MPI. The original code was made to test the performance of transport sweeps which will be subsequently described.

Tycho 2 solves the same equation with some additions. The new code adds energy dependence to the calculations to give realistic computational results for cases with several energy groups. The code is implemented in MPI and OpenMP and uses only open source software and libraries to generate meshes and compile.

2 Compiling and running the code

The code is split into the main executable named `sweep.x` and several utilities dealing with meshes in the `util` directory. Compilation of all files requires a C++11 compliant compiler, MPI, and OpenMP. Some of the utilities require extra libraries, which will be detailed.

2.1 Compiling sweep.x

Quick step guide

- Go to the project's top level directory

- Copy `make.inc.example` to `make.inc`

- Set `ASSERT` and `MPICC` in the file `make.inc`

- Type `make` to build `sweep.x`

- Or type `make -j` to build `sweep.x` in parallel

Detailed instructions for compilation can be found in the `make.inc.example` file.

2.2 Compiling the utilities

The util directory contains several utilities for creating and partitioning meshes that `sweep.x` will use. When you clone Tycho 2 from github, you should get several files in the `util` directory.

- `cube_no_mesh.hdf` – File containing a cube's geometry. The cube is a 100x100x100 cube. This can be opened by certain CAD programs to be meshed such as Salome.
  http://www.salome-platform.org/

- `cube-<N>.cgns` – An unstructured, tetrahedral mesh of `cube_no_mesh.hdf` in the CGNS file format where N is the number of cells in the mesh.
  http://cgns.github.io/

- `cube-<N>.smesh` – The same meshes as given by `cube-<N>.cgns`, just in a file format made for Tycho 2 that does not require any library dependencies to be read.

- `CreateGaussLegendre.py` – This is a Python script used to create the Gauss Legendre quadrature in the main code at `src/Quadrature.cc`.

- `CreateRegularMesh.cc` – Creates a regular Cartesian mesh and then each hexahedron is split into Tetrahedra.
  No dependencies required.

- `MoabToSerialMesh.cc` – Uses the MOAB mesh library to read in a common mesh format and convert it to the smesh file format.
  http://sigma.mcs.anl.gov/moab-library/
  Requires MOAB at the least. May also require CGNS, HDF5, NetCDF, etc for your input mesh format.

- `ParallelMeshInfo.cc` – Writes data about a pmesh.
  No dependencies required.

- `ParallelMeshToMoab.cc` – Converts the pmesh file format into a file format that the Visit visualization tool can read.
  Requires MOAB.

- `PartitionColumns.cc` – Partitions an smesh into a pmesh by columns. This requires you to specify two numbers which will the the number of partitions in the x-direction and the number of partitions in the y-direction.
  No dependencies required.

- `RefineSerialMesh.cc` – Creates a new smesh that has been refined by 8x.
  No dependencies required.

- `SerialMeshInfo.cc` – Writes data about an smesh.
  No dependencies required.

- `SerialMeshToMoab.cc` – Converts the smesh file format into a file format that the Visit visualization tool can read.
  Requires MOAB.

- `SerialToParallelMesh.cc` – Partitions an smesh using the Metis partitioning library. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview Requires Metis (just the serial version).

The general workflow is to copy `make.inc.example` to `make.inc`. Fill in specific information you need for `make.inc`. Then type `make <utility name>`. For instance, if you want to make the `SerialToParallelMesh` utility, type `make SerialToParallelMesh`. Then run `SerialToParallelMesh.x`. Every utility can be run with no arguments to get directions for use.

The one utility you will need to compile is a utility to partition an smesh into a pmesh. For this, either use `PartitionColumns` or `SerialToParallelMesh`. If you want to do a weak scaling study, you will need the `RefineSerialMesh` utility to create a series of meshes that are 8x larger than one another. If you want to visualize your mesh, you will need the `SerialMeshToMoab` or `ParallelMeshToMoab` utilities. The other utilities are mainly for development purposes.

Creating a pmesh   To run the main executable `sweep.x`, you must supply a pmesh as an argument to the executable. Several smesh files containing meshes of cubes are supplied, but it is also possible to supply your own mesh. Either the PartitionColumns or SerialToParallelMesh utility must be used to convert an smesh file into a pmesh file. PartitionColumns requires no library dependencies and SerialToParallelMesh requires the Metis partitioning library.

Creating your own mesh   If you wish to supply your own mesh, you will need to convert your mesh into an smesh. This can be done using the MoabToSerialMesh utility which requires the MOAB mesh library. MOAB can read several mesh file format, but extra libraries may have to be included in the build process for your mesh format.

Visualization   If you wish to visualize either an smesh or pmesh, you will use either the SerialMeshToMoab or ParallelMeshToMoab utilities. These require the MOAB mesh library, but no other dependencies are required. SerialMeshToMoab will create a .vtk file that can be read by Visit. ParallelMeshToMoab will create a .vtk file for each partition of the pmesh and a .visit file linking all the .vtk files together. You can then read the .visit file in Visit to visualize the mesh and its partitions.

The build process   The build process is almost the same as for the main executable.

- Copy `make.inc.example` to `make.inc`

- Set `CPP` in the file `make.inc`

- Set any of the library directories needed by the utilities you will build

- Type `make <name of utility>`. For instance `make PartitionColumns` will create the executable `PartitionColumns.x`. Typing `make` without a target will attempt to build all the utilities.

2.3   Running sweep.x

To run `sweep.x`, type `./sweep.x <.pmesh file> <input.deck file>`. This requires that you have created a pmesh as described in the previous section and you have copied `input.deck.example`

to `input.deck` (or any other name you prefer). In `input.deck`, you have the following options to set.

- `snOrder` – Order of the angular quadrature. This can be any even number from 2 to 30.

- `iterMax` – Maximum number of iterations for source iteration

- `errMax` – Tolerance for the relative error for source iteration

- `maxCellsPerStep` – Maximum number of of cell/angle pairs to compute for Ψ before communication via MPI

- `intraAngleP` – This can be 0, 1, 2, 3, or 4 for random, b-level, BFDS, DFDS, and DFHDS

- `interAngleP` – This can be 0, 1, or 2 for interleaved, globally prioritized, and locally prioritized

- `nGroups` – Number of energy group. Should be 1 or greater.

- `SweepType` Type of sweeper to use. Possible values are commented in the input.deck file.

An example `run.sh` script is supplied which will create a pmesh with 1 partition and run `sweep.x` in serial.

3 The underlying mathematics

3.1 The equation

Tycho 2 solves the following kinetic equation with isotropic scattering

$$\Omega \cdot \nabla_x \Psi(x, \Omega, E) + \sigma_t \Psi(x, \Omega, E) = \frac{\sigma_s}{4\pi} \int_{\mathbb{S}^2} \Psi(x, \Omega', E) d\Omega' + Q(x, \Omega, E). \tag{1}$$

The function Ψ is the unknown, σ_t and σ_s are the total and scattering cross sections with $\sigma_t > \sigma_s$ and are constant. The function Q is a known source.

The independent variables are:

- Space – $x \in \mathcal{D} \subset \mathbb{R}^3$,

- Direction – $\Omega \in \mathbb{S}^2$ (the unit sphere),

- Energy – $E \in \mathbb{R}^{\geq 0}$.

Notice the equation is dependent on energy, but the different energies are not coupled. This is done on purpose since the goal is to test sweeping strategies which are uncoupled in energy.

For the purpose of simpler notation, define

$$\Phi(x, E) = \int_{\mathbb{S}^2} \Psi(x, \Omega', E) d\Omega' \tag{2}$$

to get the equivalent equation

$$\Omega \cdot \nabla_x \Psi(x, \Omega, E) + \sigma_t \Psi(x, \Omega, E) = \frac{\sigma_s}{4\pi} \Phi(x, E) + Q(x, \Omega, E). \tag{3}$$

3.2 Method of discretization

Equation (3) is discretized using: discontinuous Galerkin (DG) with linear elements in x, discrete ordinates in Ω, and energy groups in E.

Energy discretization. Since equation (3) is not coupled in energy, the method of discretization of E does not matter. However, it is typical to discretize E into energy groups. Hence, the discretization in energy is denoted by E_g where g is an integer indexing the energy group (though any other discretization is fine).

$$\Omega \cdot \nabla_x \Psi_g(x, \Omega) + \sigma_t \Psi_g(x, \Omega) = \frac{\sigma_s}{4\pi} \int_{\mathbb{S}^2} \Psi_g(x, \Omega') d\Omega' + Q_g(x, \Omega). \tag{4}$$

Angle discretization. Angle is discretized via discrete ordinates, which is often denoted as S_N where N is the order of the discretization. Tycho 2 implements the Chebyshev-Legendre quadrature of the sphere. This involves a Cartesian product of N Legendre nodes on the z-axis and $2N$ equally spaced points on the circle for each Legendre node. The weights are proportional to the weights of the Legendre quadrature and scaled to sum to 4π. The nodes and weights will be denoted by Ω_q and w_q, where q stands for the quadrature index.

Equation (1) becomes

$$\Omega_q \cdot \nabla_x \Psi_{qg}(x) + \sigma_t \Psi_{qg}(x) = \frac{\sigma_s}{4\pi} \sum_{q'=0}^{2N^2-1} w_{q'} \Psi_{q'g}(x) + Q_{qg}(x). \tag{5}$$
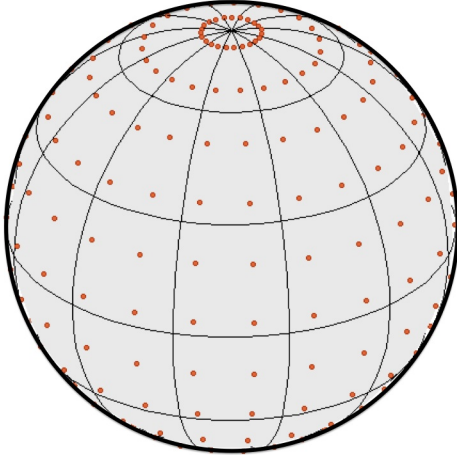
Equation (3) becomes

$$\Omega_q \cdot \nabla_x \Psi_{qg}(x) + \sigma_t \Psi_{qg}(x) = \frac{\sigma_s}{4\pi} \Phi_g(x) + Q_{qg}(x), \tag{6}$$

where

$$\Phi_g(x) = \sum_{q'=0}^{2N^2-1} w_{q'} \Psi_{q'g}(x). \tag{7}$$

Below is a depiction of the quadrature nodes on the sphere.

Spatial discretization. Since source iteration is used to converge the RHS of equation (1), it will be easier to show the equations for the simpler problem

$$\Omega_q \cdot \nabla_x \Psi_{qg}(x) + \sigma_t \Psi_{qg}(x) = Q_{qg}(x). \tag{8}$$

In this formulation, the scattering term has been absorbed into the source $Q_{qg}(x)$.

Space is discretized via linear DG. In each tetrahedral cell C_i, define basis functions $b_{ik}|_{k=0}^3$ to be linear interpolation functions with

$$b_{ik} = \begin{cases} 1, & \text{at vertex } k \text{ of cell } C_i \\ 0, & \text{at other vertices of cell } C_i \end{cases}. \tag{9}$$

Also, define

$$\Psi^h|_{C_i} = \sum_k \Psi_{ik} b_k. \tag{10}$$

Then the DG formulation is derived by starting with the equation

$$\Omega \cdot \nabla_x \Psi(x) + \sigma_t \Psi(x) = Q(x), \tag{11}$$

where the subscripts q, g are implicit. Then multiply by test function b_j in cell C_i and integrate to get

$$\int_{C_i} \Omega \cdot \nabla_x \Psi b_j dx + \int_{C_i} \sigma_t \Psi b_j dx = \int_{C_i} Q b_j dx. \tag{12}$$

Then use integration by parts on the first integral to get

$$-\int_{C_i} \Omega \cdot \nabla_x b_j \Psi dx + \sum_k \int_{F_{ik}} (\Omega \cdot \nu_k) b_j \hat{\Psi}^{(k)} dA + \int_{C_i} \sigma_t \Psi b_j dx = \int_{C_i} Q b_j dx. \tag{13}$$

Here, k indexes over the faces F_{ik} of cell i, and ν_k is the outward normal to face F_{ik}. The only thing left to define is the numerical flux $\hat{\Psi}^{(k)}$. Since Ψ is discontinuous at the faces, this is not well defined and must be defined by the numerical method. We use the upwind flux

$$\hat{\Psi}^{(k)} = \begin{cases} \Psi|_{F_{ik}} \text{ from } C_i, & \text{if } \Omega \cdot \nu_k > 0 \\ \Psi|_{F_{ik}} \text{ from adjacent cell}, & \text{if } \Omega \cdot \nu_k < 0 \end{cases} \tag{14}$$

Putting everything together, one gets the linear system below. The derivation is described in an accompanying document. Face k is defined to be opposite vertex k as shown in the figure.

$$\frac{1}{12}\bar{A}_1 \left(2\hat{\Psi}_0^{(1)} + \hat{\Psi}_2^{(1)} + \hat{\Psi}_3^{(1)}\right) + \frac{1}{12}\bar{A}_2 \left(2\hat{\Psi}_0^{(2)} + \hat{\Psi}_1^{(2)} + \hat{\Psi}_3^{(2)}\right) + \frac{1}{12}\bar{A}_3 \left(2\hat{\Psi}_0^{(3)} + \hat{\Psi}_1^{(3)} + \hat{\Psi}_2^{(3)}\right)$$
$$+ \frac{1}{12}\bar{A}_0 \left(\Psi_0 + \Psi_1 + \Psi_2 + \Psi_3\right) + \frac{\sigma_t V}{20} \left(2\Psi_0 + \Psi_1 + \Psi_2 + \Psi_3\right) = \frac{V}{20} \left(2Q_0 + Q_1 + Q_2 + Q_3\right)$$

$$\frac{1}{12}\bar{A}_0 \left(2\hat{\Psi}_1^{(0)} + \hat{\Psi}_2^{(0)} + \hat{\Psi}_3^{(0)}\right) + \frac{1}{12}\bar{A}_2 \left(\hat{\Psi}_0^{(2)} + 2\hat{\Psi}_1^{(2)} + \hat{\Psi}_3^{(2)}\right) + \frac{1}{12}\bar{A}_3 \left(\hat{\Psi}_0^{(3)} + 2\hat{\Psi}_1^{(3)} + \hat{\Psi}_2^{(3)}\right)$$
$$+ \frac{1}{12}\bar{A}_1 \left(\Psi_0 + \Psi_1 + \Psi_2 + \Psi_3\right) + \frac{\sigma_t V}{20} \left(\Psi_0 + 2\Psi_1 + \Psi_2 + \Psi_3\right) = \frac{V}{20} \left(Q_0 + 2Q_1 + Q_2 + Q_3\right)$$

$$\frac{1}{12}\bar{A}_0\left(\hat{\Psi}_1^{(0)} + 2\hat{\Psi}_2^{(0)} + \hat{\Psi}_3^{(0)}\right) + \frac{1}{12}\bar{A}_1\left(\hat{\Psi}_0^{(1)} + 2\hat{\Psi}_2^{(1)} + \hat{\Psi}_3^{(1)}\right) + \frac{1}{12}\bar{A}_3\left(\hat{\Psi}_0^{(3)} + \hat{\Psi}_1^{(3)} + 2\hat{\Psi}_2^{(3)}\right)$$

$$+ \frac{1}{12}\bar{A}_2\left(\Psi_0 + \Psi_1 + \Psi_2 + \Psi_3\right) + \frac{\sigma_t V}{20}\left(\Psi_0 + \Psi_1 + 2\Psi_2 + \Psi_3\right) = \frac{V}{20}\left(Q_0 + Q_1 + 2Q_2 + Q_3\right)$$

$$\frac{1}{12}\bar{A}_0\left(\hat{\Psi}_1^{(0)} + \hat{\Psi}_2^{(0)} + 2\hat{\Psi}_3^{(0)}\right) + \frac{1}{12}\bar{A}_1\left(\hat{\Psi}_0^{(1)} + \hat{\Psi}_2^{(1)} + 2\hat{\Psi}_3^{(1)}\right) + \frac{1}{12}\bar{A}_2\left(\hat{\Psi}_0^{(2)} + \hat{\Psi}_1^{(2)} + 2\hat{\Psi}_3^{(2)}\right)$$

$$+ \frac{1}{12}\bar{A}_3\left(\Psi_0 + \Psi_1 + \Psi_2 + \Psi_3\right) + \frac{\sigma_t V}{20}\left(\Psi_0 + \Psi_1 + \Psi_2 + 2\Psi_3\right) = \frac{V}{20}\left(Q_0 + Q_1 + Q_2 + 2Q_3\right)$$
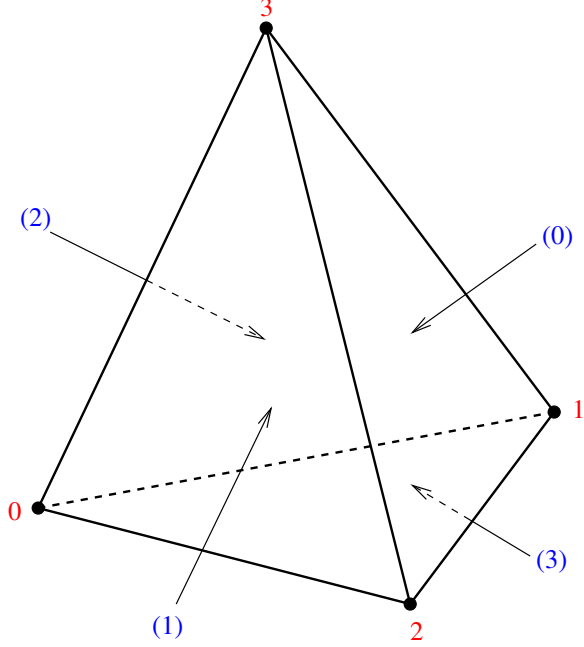

3.3   Source iteration

The overall algorithm is to lag the data from the integral on the RHS of equation (3). This is called source iteration

$$\Omega \cdot \nabla_x \Psi^{k+1} + \sigma_t \Psi^{k+1} = \frac{\sigma_s}{4\pi}\Phi^k + Q. \tag{15}$$

Tycho 2 iterates this process until $||\Phi^{k+1} - \Phi^k||_\infty / ||\Phi^{k+1}||_\infty < \text{tolerance}$.

3.4   Sweeps

The most common method of solving the discretized form of equation (11) for Ψ is called a sweep. Consider the following 2D example shown below.

Transport Direction
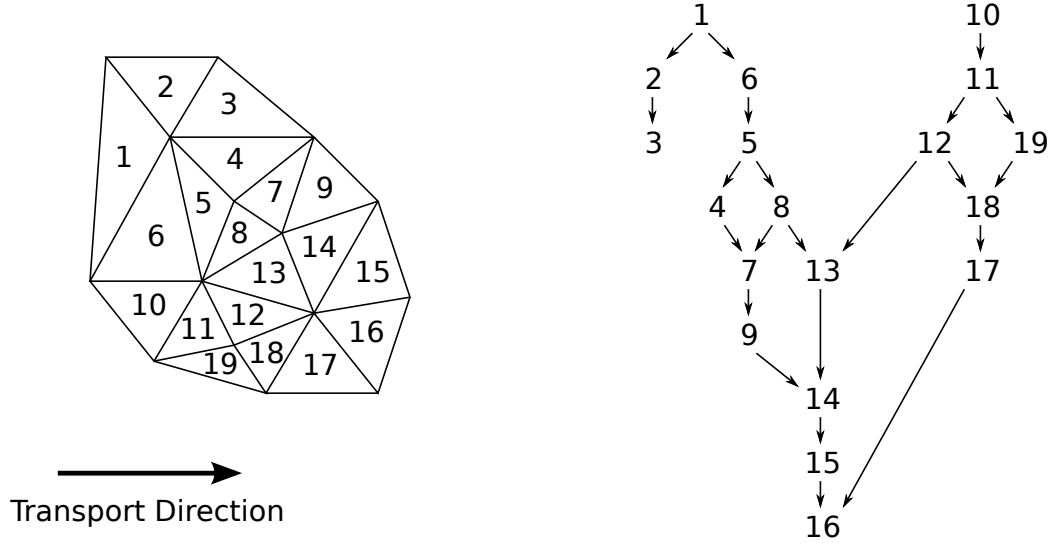
The sweep algorithm for the transport direction (angle) shown, shows that cells 1 and 10 have no dependencies except for the incoming boundary. Once Ψ is computed in these cells, the upwind flux is calculated for the boundaries between (1,2), (1,6), and (10,11). With this boundary data between cells known, Ψ in cells 2, 6, and 11 can be computed. This continues until Ψ is computed in all cells.

A sweep is needed for each angle and energy group. Fortunately, all these sweeps are completely independent and therefore trivially parallelizable.

3.4.1 Sweeps Algorithms

Several sweep algorithms have been employed. The input deck currently allows 4 types of sweeps: `OriginalTycho1`, `OriginalTycho2`, `GraphTraversal`, and `PBJ`. All but the `PBJ` algorithm rely on each cell-angle pair having a priority assigned. To understand why, consider the same sweep problem as shown in the previous figure but now split via MPI into 2 partitions.
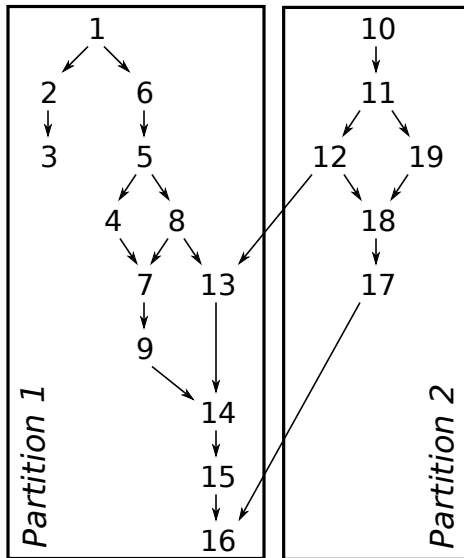
For partition 2, cell 10 must be computed first, then cell 11. After this, there is a choice. Either cell 12 or 19 can be computed, but cell 12 is probably best so that data can be communicated to compute cell 13 in partition 1. Generally, the cell-angle priorities will attempt to get data to a

partition boundary as quickly as possible. The formulas for the priorities are detailed below but an explanation of the reasoning for each priority may be found in [1].

3.4.2 Intra-Angle Priorities

Priorities are first created cells in each angle separately. This is the definition of the intra-angle priorities. Fundamental to all the priority calculations is the notion of B-Level of a cell. This is defined to be the longest distance to leaf of the *entire* graph (not just for the partition). For instance, node 17 has a B-Level of 1, node 16 has a B-Level of 0, and node 10 has a B-Level of 6 by traversing cells: 10, 11, 12, 13, 14, 15, 16. Leaves of a graph have a B-Level of zero.

Random   The random priority assigns a uniformly random priority to all cells.

B-Level   The b-level priority assigns to each cell a priority equal to the cell's b-level.

BFDS   This stands for Breadth First Descendant Seeking. Each cell's priority is equal to the highest b-level of any descendant in another partition. If there is no descendant in another partition, the priority is set to zero.

DFDS   This stands for Depth First Descendant Seeking. Each cell with a child in another partition has a priority equal to the maximum b-level of such children plus a constant equal or greater than the number of levels of the graph. Parents are recursively assigned a priority of one less than the maximum of its children. If a cell has no descendants in another partition, its priority is set to zero.

DFHDS   This stands for Depth First Highest Descendant Seeking. Each cell with a child in another partition has a priority equal to the maximum b-level of such children multiplied by (rather than addition as in DFDS) a constant equal or greater than the number of levels of the graph. The rest of the algorithm is the same as for DFDS.

3.4.3 Inter-Angle Priorities

For a given cell, a priority is assigned for each angle independently via one of the intra-angle priorities above. Now these priorities must be combined.

Interleaved   For this priority type, the inter-angle priority is set to the intra-angle priority. This can cause angles to be interleaved when cell-angle pairs are sorted by priority.

Globally Prioritized   A sufficiently large constant is multiplied by the angle index to ensure a cell-angle pair with angle index less than another angle index has a lower priority.

Locally Prioritized   First, for each angle, the maximum priority across cells is calculated to created an array of maximum priorities (with size of the number of angles). The max priority array is sorted and angle indices are reordered with respect to the sorted max priority array. Then the globally prioritized algorithm is used.

3.4.4 The Sweep Algorithms

The general sweep algorithm works as follows.

```
Assign priorities for each cell-angle pair
Calculate number of dependencies (parents) for each cell-angle pair
Place all cell-angle pairs with no dependencies in a priority queue
While (Still some cell-angle pair to compute)
  For i = 1 to maxCellsPerStep
    Pop cell-angle pair off priority queue
    Calculate Psi on cell-angle pair
    Update number of dependencies for children on MPI rank
    If (child has no dependencies)
      Put child in priority queue
    Send/Recv MPI rank's boundary data
    Update dependencies
    Place any cell-angle pairs with no dependencies in priority queue
```

OriginalTycho 1 and 2   Both of these sweep types use the same overall algorithm but differ on their OpenMP implementation. In this section, the OpenMP implementation will not be discussed, and the text below equally describes both sweep types.

For this sweep type, the schedule of updates for Ψ on each cell-angle pair is precomputed. The for loop in the algorithm above is called a step and a queue of computational tasks is initially created for that loop. Then, when it comes time to do the sweep, tasks are popped off the queue for each step.

GraphTraversal   This sweep algorithm follows the above algorithm exactly. In particular, the dependencies are updated calculated in real time for the sweep as opposed to precomputing them like the OriginalTycho algorithms.

PBJ   PBJ stands for Parallel Block Jacobi. In this algorithm, every MPI rank has an initial guess for the boundary data and a sweep is performed for each partition without regard for neighboring partitions. Then boundary data is communicated between all MPI ranks and sweeps are performed on each individual partition again. This is iterated until Ψ does not change.

References

[1] Shawn D. Pautz, "An Algorithm for Parallel S$_n$ Sweeps on Unstructed Meshes," *Nuclear Science and Engineering*, 140, 111-136 (2002).